Système de Gestion de Base de Données (SGBD)

Introduction

Introduction

Une base de données (BD) est un ensemble d'informations archivées dans des mémoires accessibles à des ordinateurs en vue de permettre le traitement des diverses applications prévues pour elles.

Introduction

L'intérêt d'une BD est de regrouper les données communes à une application dans le but:

- d'éviter les redondances et les incohérences qu'entraînerait fatalement une approche où les données seraient réparties dans différents fichiers sans connexions entre eux,
- d'offrir des langages de haut niveau pour la définition et la manipulation des données,
- de partager les données entre plusieurs utilisateurs,
- de contrôler l'intégrité, la sécurité et la confidentialité des données,
- D'assurer l'indépendance entre les données et les traitements.

Introduction

Les bases de données sont gérées par des logiciels spécialisés appelés systèmes de gestion de base de données (SGBD).

Les fonctions des SGBD:

DEFINITION DES DONNEES

⇒Langage de définition des données (DDL)

(conforme à un modèle de données)

MANIPULATION DES DONNEES

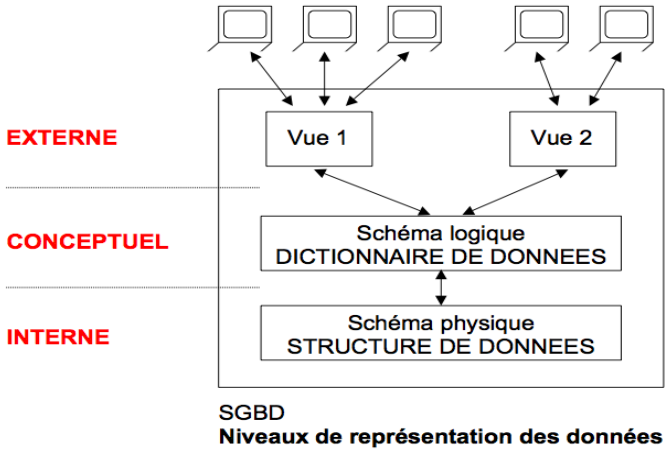
Interrogation

Mise à jour (insertion, suppression, modification)

⇒Langage de manipulation des données (DML)

(langage de requête déclaratif)

Architecture ANSI/SPARC



Niveaux de représentation des données

Schéma conceptuel

Le schéma conceptuel est une représentation du monde réel auquel se rapporte la BD.

Principaux concepts :

- Entité (ou objet) : une personne, un livre
- Propriété (ou attribut) : le titre d'un livre, l'adresse d'une personne
- Association : entre une personne et le livre dont elle est l'auteur
- Agrégat : une adresse composée d'une rue et d'un code postal
- Collection : un ensemble de personnes, une liste de prénoms

Principaux modèles conceptuels

- **1ère** génération:
  - Hiérarchique
  - Réseau
- **2e** génération:
  - Relationnel
  - Entité-Association
- **3e** génération:
  - Modèle orienté objet
  - Modèle objet-relationnel
  - UML
- **4e** génération:
  (les données du Web)
  - XML

Exemple

- ✓ Soit une BD décrivant les livres d'une bibliothèque et leurs auteurs.
- ✓ On suppose qu'un livre est identifié par son code et un auteur par son nom.
- ✓ On se place à l'instant ou la bibliothèque contient:
  - Un seul livre,
  - Ayant le code BD/46 et le titre 'les BD',
  - Écrit par Jean Dupont né en 1960 et Pierre Durand né en 1953.

Exemple en relationnel

Schéma

Livre (code,titre)

Auteur(nom, prenom, annee_nais)

Liv_aut(code,nom)

Extension

Livre

code	titre
BD/46	Les BD

Auteur

nom	prenom	Annee_nais
Dupont	Jean	1960
Durand	Pierre	1953

Liv_aut

code	nom
BD/46	Dupont
BD/46	Durand

Exemple en orienté objet

Exemple en orienté objet

Schéma

```
Classe livre
    Attribut code: text
    Attribut titre: texte
    Attribut ecrit_par: liste (auteur)
Classe auteur
    Attribut nom: texte
    Attribut prenom: texte
    Attribut annee_nais: entier
    Méthode age(): entier {annee_courante-self-> annee_naiss}
Livre: ensemble (livre)
Auteurs: ensemble (auteur)
```

Variables persistantes

```
Livres={l1}

Auteurs={a1,a2}
```

Instances

```
(l1,{code= "BD/46", titre= "Les BD", ecrit_par=[a1,a2]})

(a1,{nom="Dupond", prenom="Jean",annee_nais=1960})

(a2,{nom="Durand", prenom= »Pierre",annee_nais=1953})
```


Exemple en XML

Fichier DTD

```
<!ELEMENT bd (livres, personnes)
<!ELEMENT livres (livre*)>
<!ELEMENT livre (cote, titre, écrit_par*)>
<!ELEMENT cote (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT écrit_par EMPTY>
<!ATTLIST écrit_par ref IDREF>
<!ELEMENT auteurs (personne*)>
<!ELEMENT auteur (nom, prénom,
année_naissance)>
<!ATTLIST auteur id ID>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT année_naissance (#PCDATA)>
```

Fichier XML

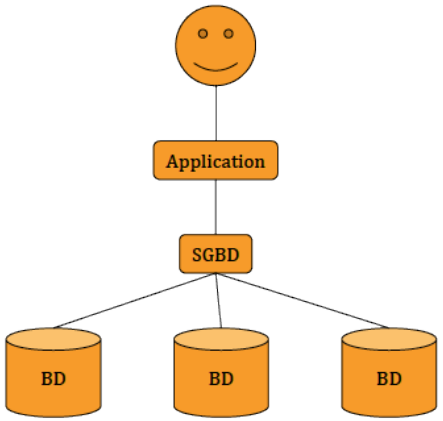
```
<bd>
<livres>
<livre>
<cote>BD/46</cote>
<titre>Les BD en BD</titre>
<écrit_par    ref="a1"/
><écrit_par ref="a2"/>
</livre>
</livres>
<auteurs>
<auteur id="a1">
<nom>Dupont</nom>
<prénom>Jean</prénom>
<année_naissance>1960</
année_naissance>
</auteur>
<auteur id="a2">
<nom>Durand</nom>
<prénom>Pierre</prénom>
<année_naissance>1953</
année_naissance>
</auteur>
</auteurs>
</bd>
```


Architecture d'un SGBD

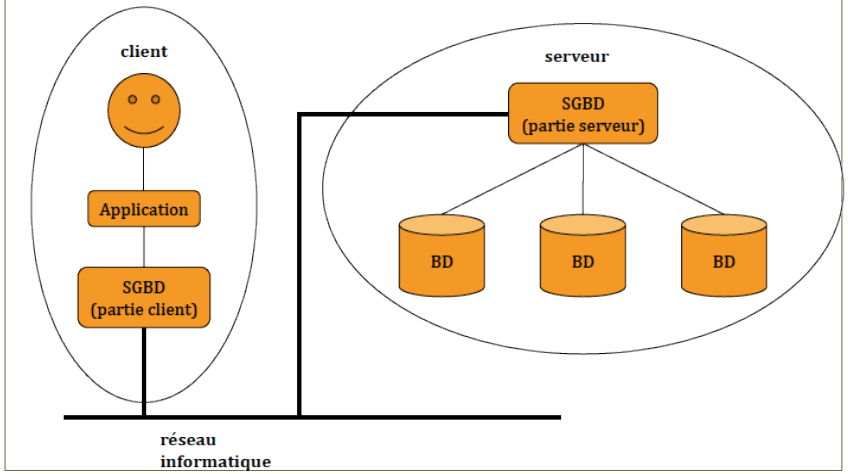
On distingue 3 grands types d'architecture :

- ✓ Architecture centralisée,
- ✓ Architecture client-serveur,
- ✓ Architecture trois-tiers..

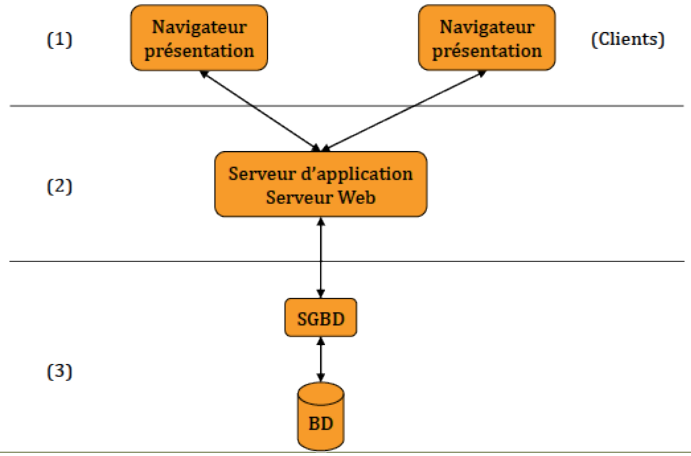
Architecture centralisée



Architecture client-serveur



Architecture trois tiers



Langage de base de données

- C'est au travers d'un langage déclaratif sont réalisées la définition et la manipulation d'une BD.
- Le plus connu et le plus utilisé de ces langages est SQL.
- Modes d'utilisation :
  - **Intégré** dans un langage hôte (C, Java…),
  - **Autonome**.

Contrôles

- **Intégrité**

Les données stockées dans une BD doivent respecter un certain nombre de contraintes dites d'intégrité. Un SGBD doit assurer qu'elles sont toujours respectées.

- **Confidentialité**

Un SGBD doit permettre d'interdire à certaines personnes de réaliser certaines opérations sur une partie ou sur toute la BD.

- **Concurrence**

Plusieurs utilisateurs se partagent la même BD. Un SGBD doit gérer les conflits qui peuvent se produire lorsqu'ils manipulent simultanément les mêmes données de façon à ce que la BD ne soit pas mise dans un état incohérent.

- **Reprise après panne**

Après une panne, qu'elle soit d'origine logicielle ou matérielle, un SGBD doit être capable de restaurer la BD dans un état cohérent, le même ou le plus proche de celui précédant la panne.

Transactions

- Pour un SGBD l'unité de traitement est la transaction.
- Une transaction est un fragment de programme qui fait passer une BD d'un état cohérent à un autre état cohérent, par une suite d'actions élémentaires.
- Un exemple classique de transaction est l'opération qui transfère une somme S d'un compte bancaire A à un compte bancaire B :

```
début transaction
    solde(A) = solde(A) - S
    solde(B) = solde(B) + S
fin transaction
```

- Il est clair que cette opération ne doit pas être interrompue entre le débit de A et le crédit de B.

Indépendance données-traitements

- L'indépendance données-traitements est indispensable pour pouvoir faire évoluer facilement l'organisation logique ou physique d'une BD ou bien l'architecture matérielle du SGBD qui la gère.
- L'indépendance données-traitements permet si elle est atteinte :

  - de modifier l'organisation physique (par exemple ajouter un index pour un accès plus rapide) sans modifier le schéma conceptuel ou les programmes d'applications,

  - de modifier le schéma conceptuel (par exemple ajouter un nouveau type d'entité ou d'association) sans modifier les programmes d'applications non concernés par cet modification.

- On parle aussi **d'indépendance logique** et **d'indépendance physique**.

SQL (Structured Query Language)

Le SQL comporte 5 grandes parties:

- LDD (DDL : " Data Definition Language ") : permet de créer des bases de données, des tables, des index, des contraintes.

  Exp : CREATE, ALTER, DROP…

- LMD (DML : " Data Manipulation Language ") : permet de manipuler les données.

  Exp : INSERT, UPDATE, DELETE…

- LID (DQL : " Data Query Language ") : permet d'extraire et interroger des données.

  Exp: SELECT…

SQL (Structured Query Language)

- LCD (DCL : “ Data Control Language ”) : permet de gérer les droits d'accès aux tables.

  Exp : GRANT, REVOKE

- LCT (TCL : “ Transaction Control Language ”) : permet de contrôler la bonne exécution des transactions.

  Exp : COMMIT, ROLLBACK…

- SQL intégré : “ Embedded SQL ” : éléments procéduraux que l'on intégré à un langage hôte.

  Exp : SET, DECLARE CURSOR, OPEN, FETCH...

SQL: LDD

- **Création d’une table**

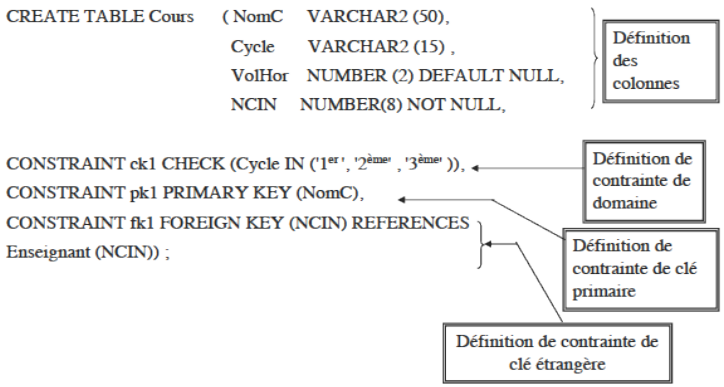
Syntaxe:

```
CREATE TABLE table (colonne1 type1 [NOT NULL|NULL],
                    colonne2 type2 [NOT NULL|NULL],
                    . . .
[CONSTRAINT nom_contrainte PRIMARY KEY (liste_attributs_clé primaire)
|NOT NULL immédiatement après la déclaration de l’attribut
|CHECK (condition) après la déclaration de l’attribut
|UNIQUE après la déclaration de l’attribut
|FOREIGN KEY (clé étrangère)
        REFERENCES nom_table (liste_colonnes)] );
```


SQL: LDD

- **Création d’une table**

Exemple:



SQL: LDD

- **Suppression d’une table**

Syntaxe:

```
DROP TABLE [IF EXISTS] nom_table[, nom_table ] ...
```

- **Modification d'une table**

Syntaxe:

```
ALTER TABLE nom_table ADD/MODIFY/CHANGE/DROP nom_colonne
```

Remarque : On ne peut pas:

- *Modifier une colonne que si la colonne ne contient que des valeurs nulles ou si la nouvelle définition est compatible avec les valeurs déjà entrées dans cette colonne.*
- *Supprimer une colonne référencée par une clé étrangère.*

SQL: LDD

- **Modification d'une table**

Exemple

```
ALTER TABLE Enseignant ADD email VARCHAR(30)

ALTER TABLE Enseignant MODIFY email VARCHAR(50)

ALTER TABLE Enseignant CHANGE email courriel VARCHAR(50)

ALTER TABLE Enseignant DROP courriel
```


SQL: LDD

- **Renommer une table**

Syntaxe:

```
RENAME ancienNom TO nouveauNom
```

Ou bien Commande équivalente :

```
ALTER TABLE ancienNom RENAME TO nouveauNom
```

Oracle permet aussi d'invalider des contraintes

```
ALTER TABLE table
{DISABLE | ENABLE} CONSTRAINT nom-contrainte
```


SQL: LMD

- **INSERT:**

Permet d'insérer des données dans une table.

Syntaxe:

```
INSERT [INTO] nom_table [(liste_des_colonnes_visées, …)]
{VALUES (liste_des_valeurs,…)}
```

Ou bien:

```
INSERT INTO table [(colonne1, colonne2,…)]
SELECT…
```

Remarque : La liste des colonnes est optionnelle; par défaut, toutes les colonnes sont dans l'ordre donné lors de la création de la table.

Exemple :

```
INSERT INTO Enseignant (nomE, prenomE) VALUES ('Ben bechir', 'Ali')
```


SQL: LMD

- **UPDATE:**

Permet de modifier des données dans une table.

Syntaxe:

```
UPDATE nom_table
       SET colonne = valeur [,colonne = valeur]
       [WHERE clause]
```

ou bien:

```
UPDATE table [ synonyme ]
SET (colonne1, colonne2, …) = (select …)
[ WHERE prédicat ]
```

Exemple :

```
UPDATE Enseignant SET nomE, prenomE VALUES ('Ben bechir', 'Ali')
```

SQL: LMD

- **DELETE**

Syntaxe:

Permet de supprimer des données dans une table.

```
DELETE [FROM] nom_table
       [WHERE clause]
```

Remarque: s'il n'y a pas de clause WHERE, toutes les lignes sont supprimées

Exemple :

```
DELETE FROM enseignant

WHERE statut = 'Contractuel';
```